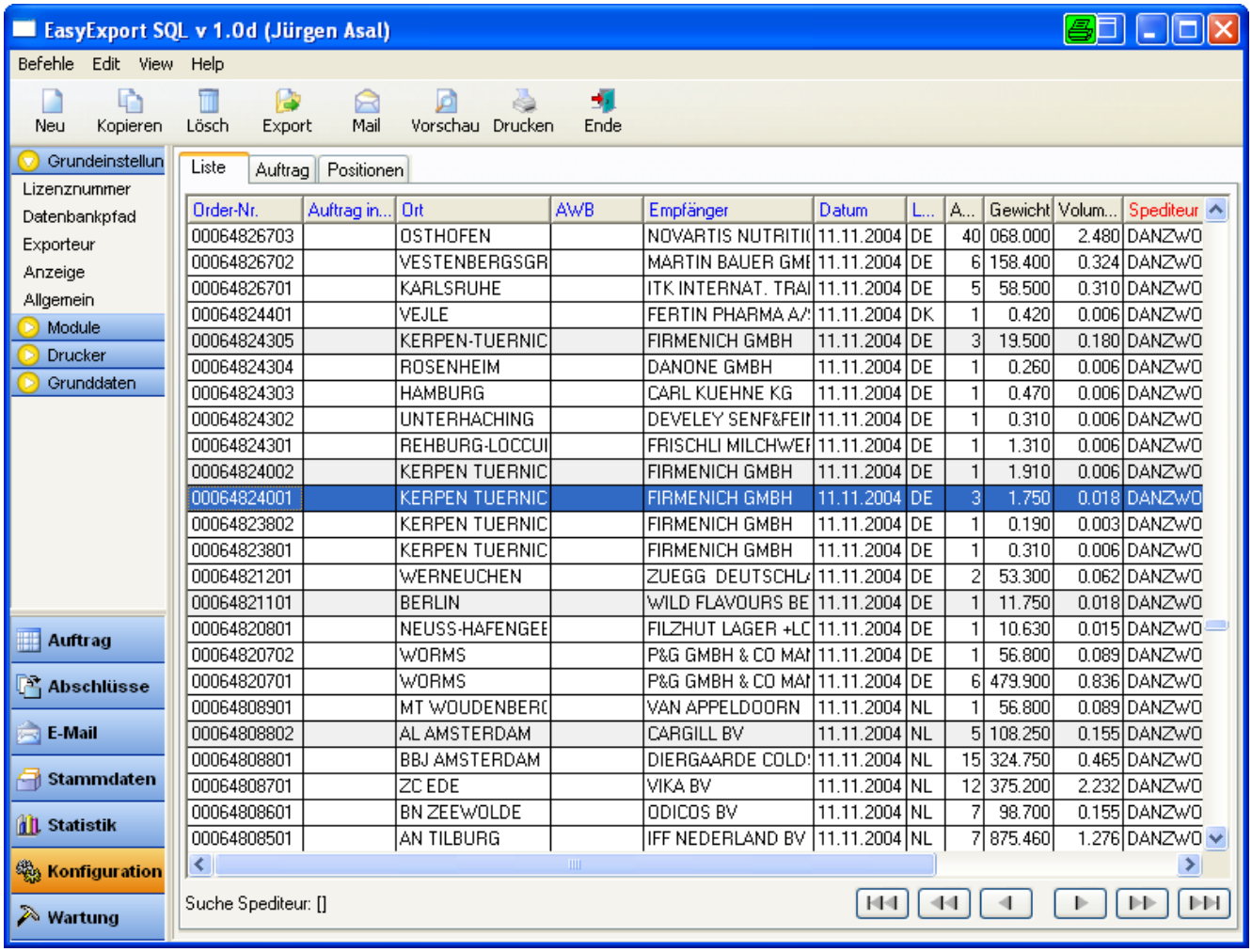
1. Ziel

Ziel ist, eine Applikation zu erstellen, welche sowohl als herkömmliches VO-Programm als auch als WEB-Applikation laufen kann.

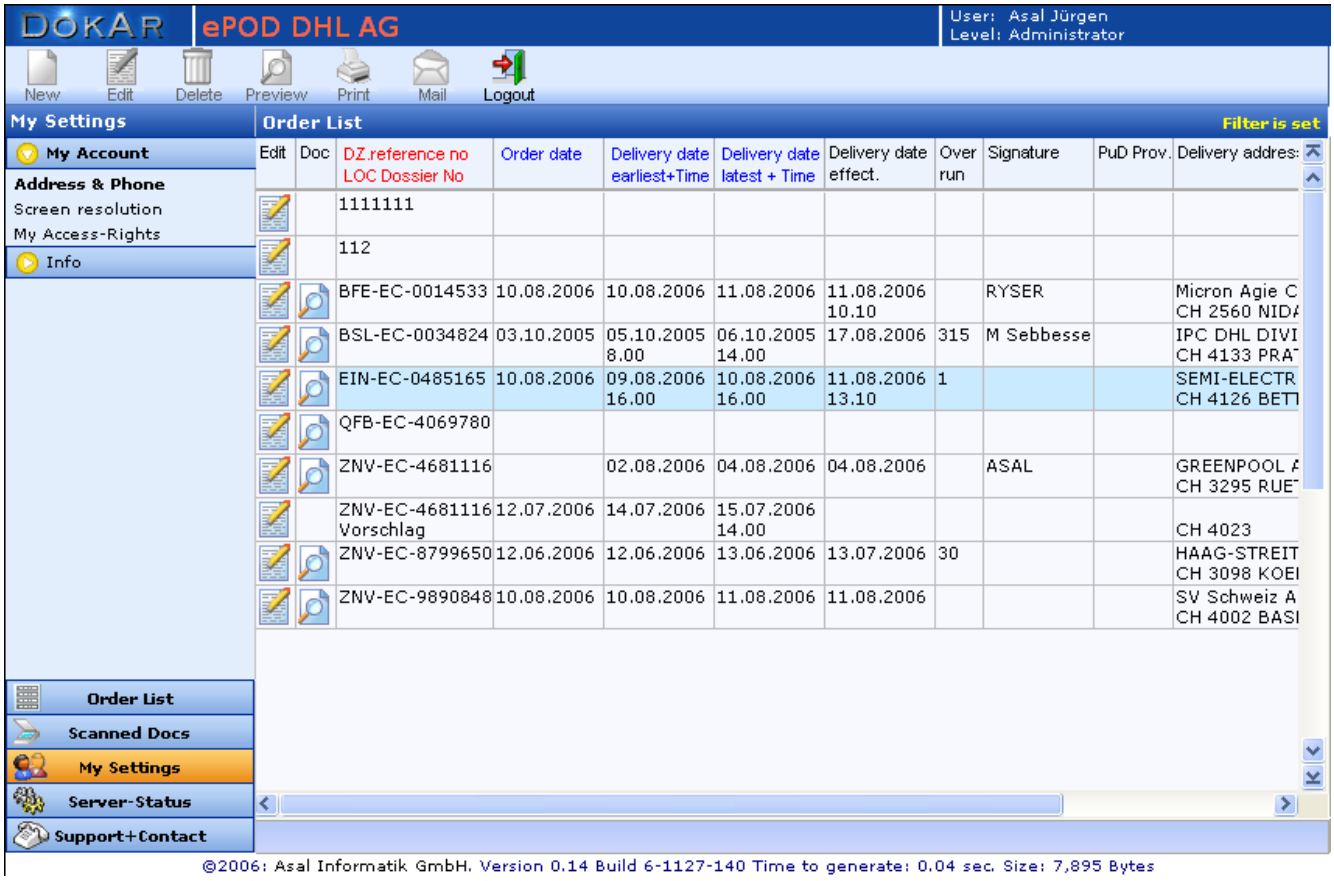
Dabei sind folgende Vorgaben zu berücksichtigen:

- Programmcode soweit möglich identisch
- Benutzeroberfläche soweit möglich identisch
- Sicherheit gegen Datendiebstahl
- Kein Installation auf Clientseite nötig

1.1. Screenshot Variante „FAT-Client“



1.2. Screenshot Variante „Internet“



2. Mehrbenutzer-Techniken im Überblick

2.1. Mehrbenutzer-Techniken

2.1.1. Netzwerk

Mit der ADS-Datenbank kann eine Datenbank mit dem TCP/IP-Protokoll angesprochen werden, welche auf einem Internetserver liegt. Dazu muss auf jedem Client eine Applikation installiert werden sowie der Firewall geöffnet werden.

Vorteil:

- Hohe Geschwindigkeit
- „Herkömmliche“ Programmiertechnik möglich

Nachteil:

- Client-Installation nötig
- Firewall-Konfiguration nötig
- Keine zentrale Verwaltung

2.1.2. Metaframe / Cytrix

Dia Applikation läuft auf dem Server. Der Bildschirm wird mit einem „Terminal“-Protokoll auf den Client übertragen.

Vorteil:

- Sehr hohe Geschwindigkeit
- „Herkömmliche“ Programmiertechnik möglich
- Zentrale Verwaltung

Nachteil:

- Client-Installation nötig
- Firewall-Konfiguration nötig
- Extrem hohe Belastung des Servers
- Nur relativ wenig User pro Server möglich (max. 100)
- Cytrix-Server ist bei hoher Belastung relativ instabil, regelmässiger Reboot nötig

2.1.3. HTML-Applikation

Dia Applikation läuft auf dem Server. Als Client dient ein Internet-Browser.

Vorteil:

- Keine Client-Installation nötig
- Zentrale Verwaltung
- Keine Firewall-Konfiguration nötig
- Niedrige Belastung des Servers
- Sehr viele User möglich

Nachteil:

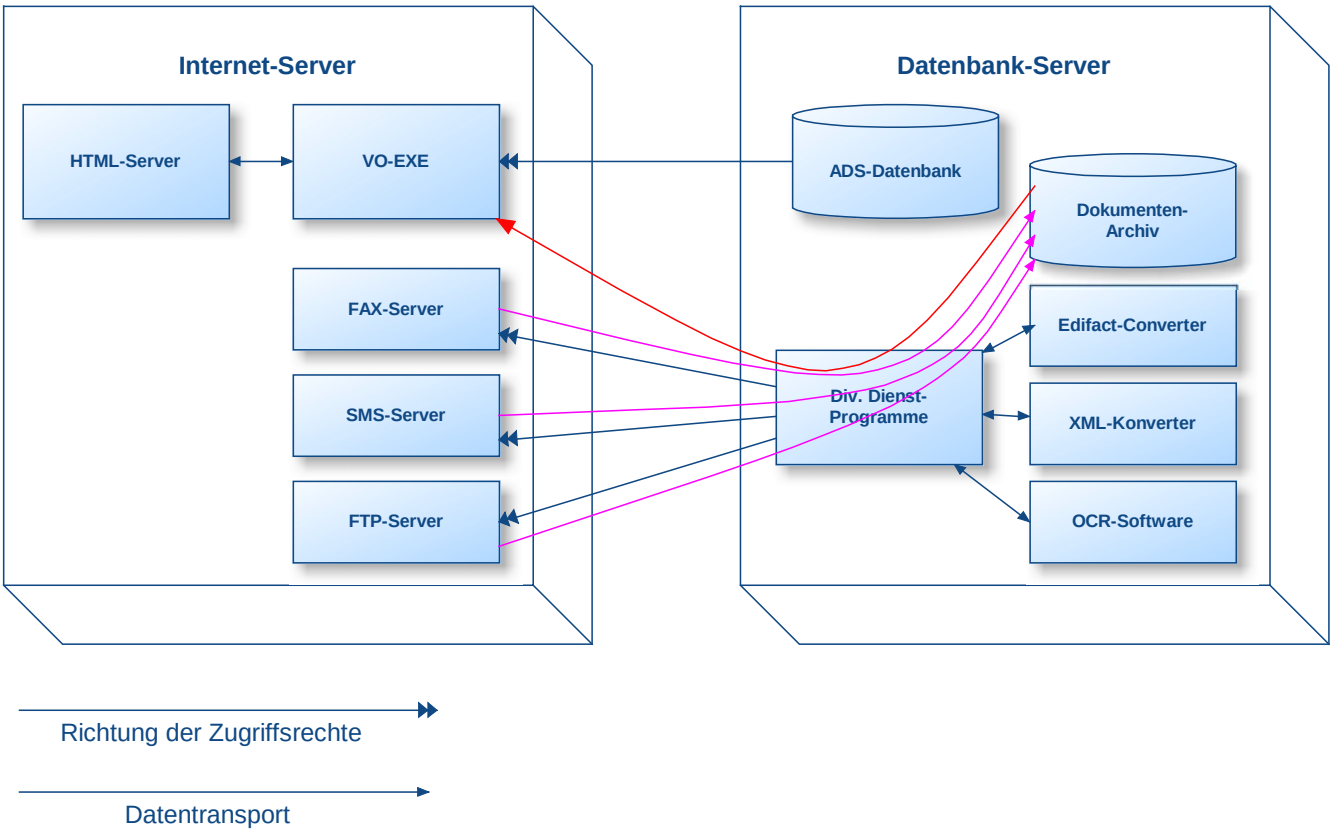
- Relativ langsam
- Neue Programmiertechnik nötig
- Kompatibilitätsprobleme mit verschiedenen Browsern

2.2. Meine Anforderungen an die Internetapplikation:

- Browserbasierendes Datenbanksystem erstellen
- Kompatibilität mit allen (halbwegs) modernen Browsern
- Kein Java, Javascript, Aktiv-X oder Plugins (Flash etc.)
- Windows, Unix und MAC-tauglich
- Etwa 35 verschiedene Dialog-Sprachen
- Bestehende VO-Funktionen müssen weiterverwendet werden können

2.3. Sicherheitskonzept

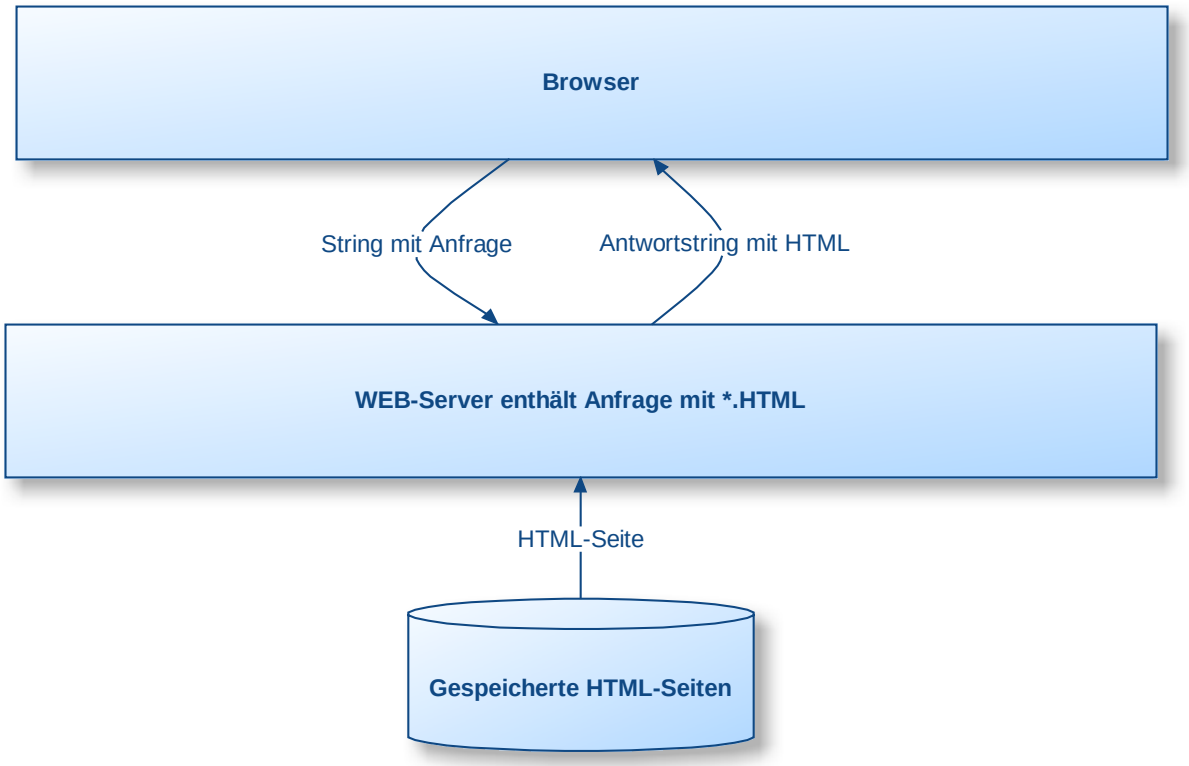
Alle im Internet sichtbaren Programmteile sind auf einem eigenen Server abgelegt. Die Daten liegen auf einem zweiten Server, welcher keinerlei Freigaben besitzt.



- Es gibt nur Freigaben auf dem Internetserver, aber keine Freigabe auf dem Datenbankserver
- Der Zugriff auf die Datenbank erfolgt nur über das TCP/IP-Protokoll, eine Freigabe ist nicht nötig
- Auf dem Internetserver sind keine Daten gespeichert, ausser denjenigen, welche ein Benutzer gerade angefordert hat
- Daten, welche via Upload, Fax oder SMS auf den Internetserver geladen werden, werden innert Sekunden auf den Datenbankserver verschoben.

2.4. WEB-Techniken

2.4.1. Kommunikation zwischen Web-Server und Browser



Die Webseiten, welche vom Server generiert werden, und die Antworten des Browsers sind nichts anderes als gewöhnliche Strings!

Aber woher nimmt der Webserver die auszuliefernden Seiten?

2.4.2. Statische HTML-Seiten

Diese Seiten sind unveränderlich auf dem Server gespeichert.

2.4.3. Dynamische HTML-Seiten

Diese Seiten werden serverseitig mittels geeigneter Technik für die jeweilige Anfrage generiert. Dabei wird normalerweise immer eine vollständige Seite übertragen.

2.4.4. Pseudo-dynamische HTML-Seiten

Die Seiten werden statisch oder dynamisch generiert und an den Browser übermittelt, dort aber mittels lokaler Programmiersprachen (JavaScript, Java, Flash etc.) dynamisch verändert.

Vorteil:

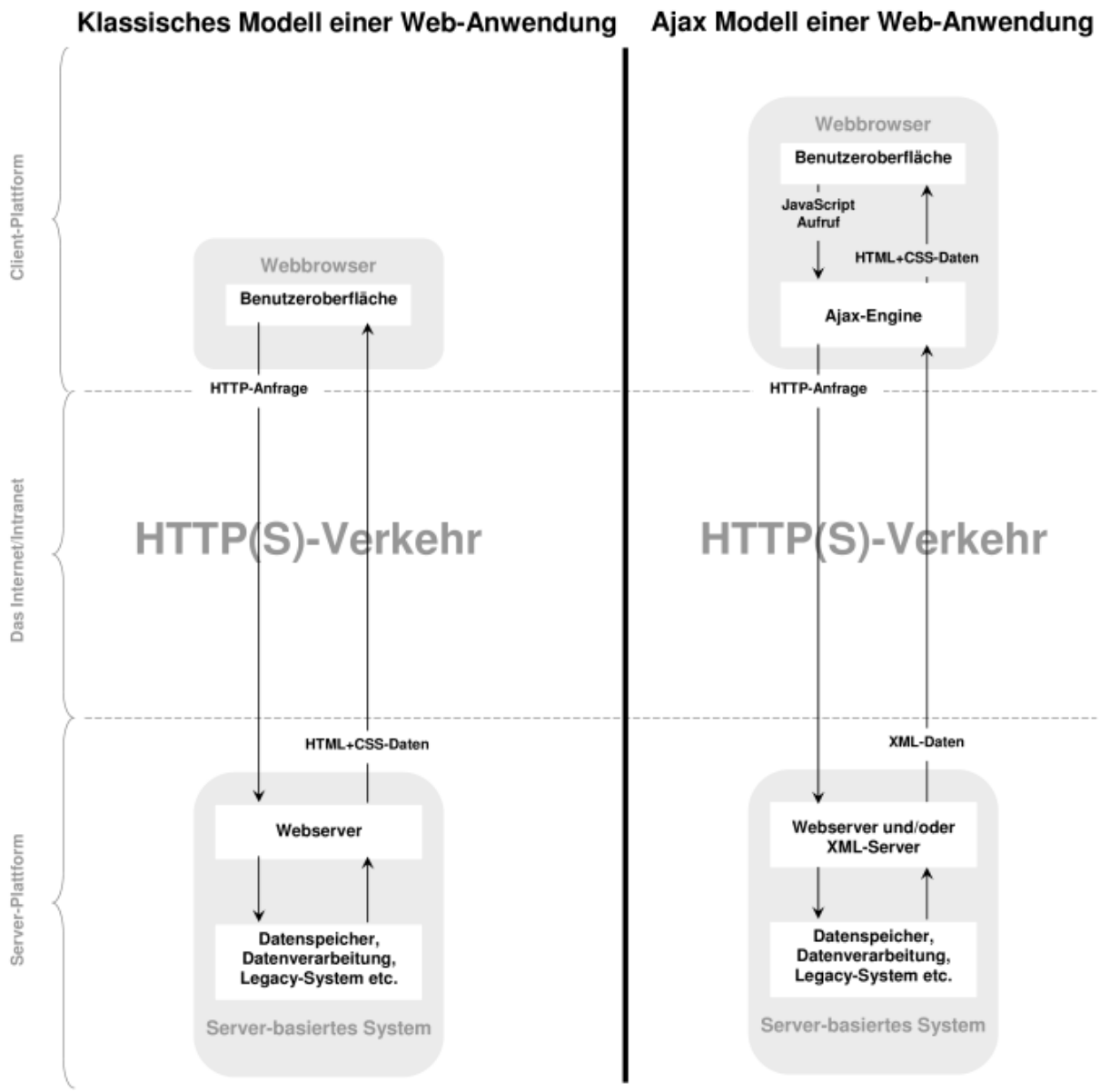
- Änderungen der Seite sind offline möglich

Nachteile:

- Der Server weiss nichts von den lokalen Änderungen
- JavaScript kann ausgeschaltet sein resp. die Programmiersprache Java kann nicht installiert seine resp. das Flash-Plugin ist möglicherweise nicht installiert

2.4.5. AJAX

Ajax ist keine Programmiersprache, sondern eine Technik, um Teile einer Seite „nachladen“ zu können.



Grafik aus Wikipedia

Beispiel eines AJAX-Codes (aus Wikipedia)

```
var xmlHttp = false;
// Mozilla, Opera, Safari sowie Internet Explorer 7
if (typeof XMLHttpRequest != 'undefined') {
    xmlHttp = new XMLHttpRequest();
}
if (!xmlHttp) {
    // Internet Explorer 6 und älter
    try {
        xmlHttp  = new ActiveXObject("Msxml2.XMLHTTP");
    } catch(e) {
        try {
            xmlHttp  = new ActiveXObject("Microsoft.XMLHTTP");
        } catch(e) {
            xmlHttp  = false;
        }
    }
}
if (xmlHttp) {
    xmlHttp.open('GET', 'beispiel.xml', true);
    xmlHttp.onreadystatechange = function () {
        if (xmlHttp.readyState == 4) {
            alert(xmlHttp.responseText);
        }
    };
    xmlHttp.send(null);
}
```

Vorteile:

- Schnellere Reaktionszeit, weil nicht eine gesamte Seite übertragen werden muss

Nachteile:

- Setzt zwingend JavaScript voraus
- Für den IE muss ein Activ-X Modul installiert werden
- Läuft nur mit modernen Browsern
- Sehr hoher Programmieraufwand
- Unterschiedliche Programmierung für Standard-konforme Browser und Microsoft-Browser
- Site ist bei deaktiviertem JavaScript, IE mit ausgeschaltetem Activ-X oder bei älteren Browsern völlig unbrauchbar
- Zur Zeit herrscht ein Wildwuchs an AJAX-Implementationen (es gibt zur Zeit über 100 verschiedene Implementationen), keinerlei Standards vorhanden

2.5. WEB-Programmiersprachen und –Techniken

2.5.1. Browserseitige Sprachen:

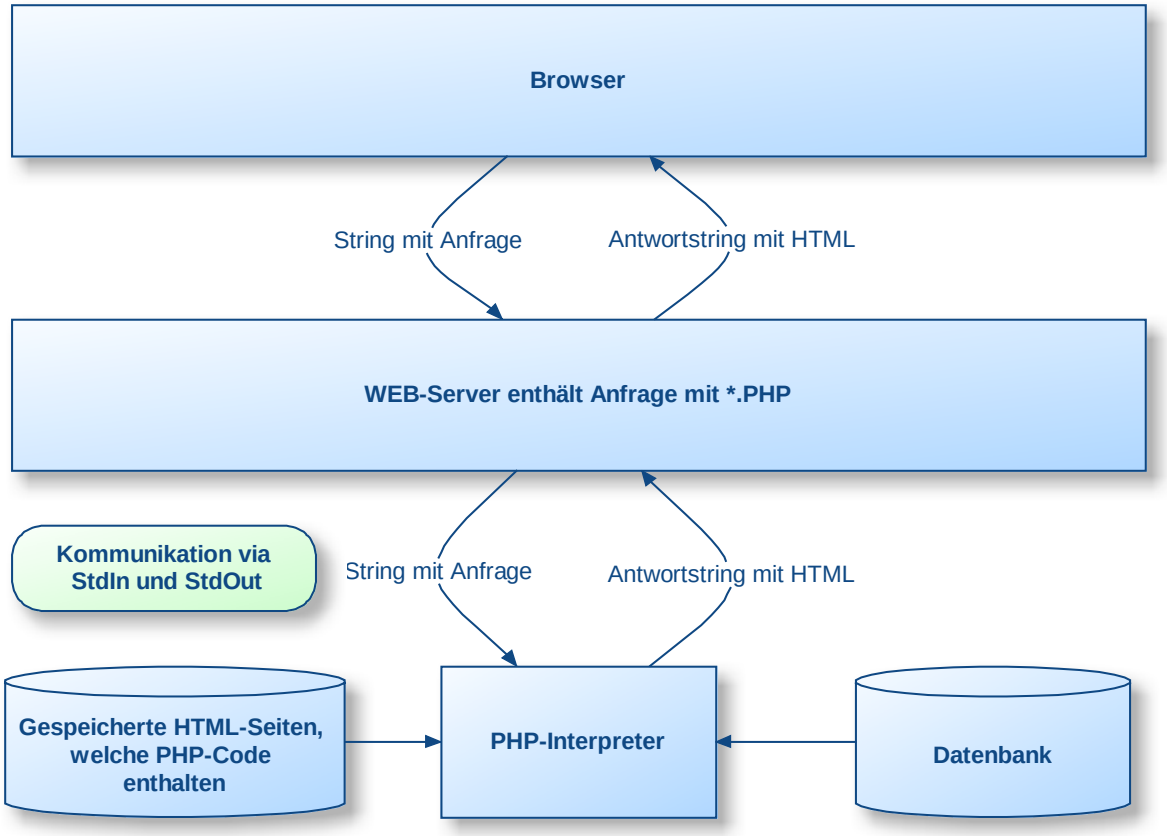
Java	• Langsam • Sicherheitsproblem • „aus der Mode gekommen“ • Umfangreiche Installation nötig
Javascript	• Wird sehr häufig für Spam missbraucht, deshalb oft ausgeschaltet • Sehr unterschiedliche Implementation gewisser Befehle in den verschiedenen Browsern
Aktiv X	• Extremes Sicherheitsrisiko • Läuft nur auf Windows-PC und Internet Explorer
Flash	• Verlangt Download und Installation eines PlugIns • Langsam (wegen Übertragung grosser Datenmengen)

2.5.2. Serverseitige Sprachen:

Pearl, PHP, Java etc.	• Nur Scriptsprachen, werden interpretiert • Grosse Projekte schwer zu handeln • Häufiger Wechsel der Modetrends, nichts langfristiges • Hohes Sicherheitsrisiko durch Sicherheitslücken • Edifact-Converter und Reportgenerator müssen zusätzlich gekauft und installiert werden
ASP, ASP.NET	• Sehr leistungsfähig • Komplex, schwierig zu erlernen • Microsoft, läuft nur auf MS-Plattformen • Grosse Inkompatibilitäten beim Versionsupdate • Edifact-Converter müssen zusätzlich gekauft und installiert werden
Visual Objects	• Alles bereits vorhanden und kann weiterverwendet werden • Akzeptanzprobleme beim Kunden • Performance nicht allzu hoch

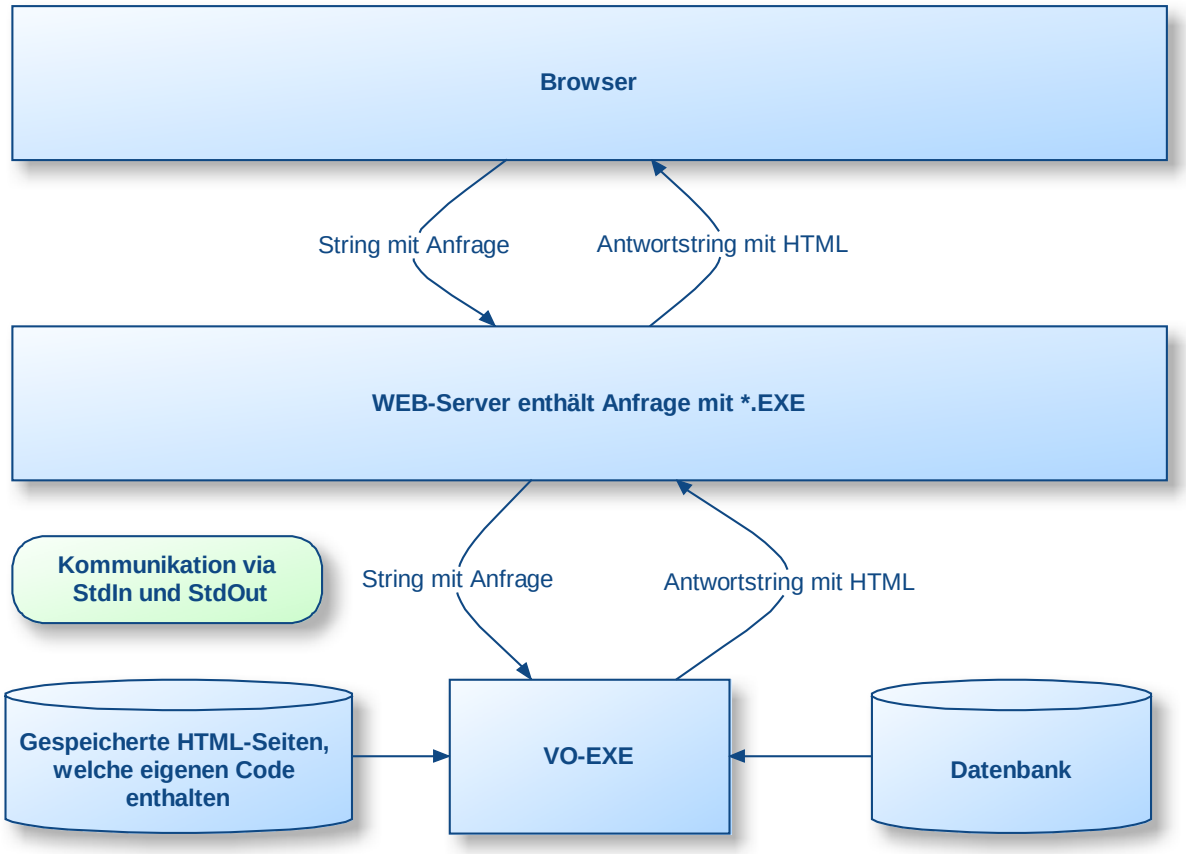
2.6. Ablauf Serverseitig mit PHP, Pearl etc.

2.6.1. Beispiel mit PHP



1. Zunächst startet ein Client (Browser) eine Anfrage für eine PHP-Datei. Dies geschieht über die Eingabe einer URL oder durch anklicken eines Links. Die Anfrage wird über das Internet an den entsprechenden Server weitergeleitet.
2. Der Server (Webserver) nimmt die Anfrage entgegen und lädt die PHP-Datei von seiner Festplatte.
3. Anschließend übergibt der Webserver den Anfrage-String an den PHP-Interpreter.
4. Der PHP-Interpreter arbeitet das PHP-Skript ab.
5. Anschließend gibt der Interpreter das Ergebnis seiner Arbeit (meist in Form einer HTML-Datei, aber auch Bilder oder PDF-Dokumente sind möglich) an den Webserver zurück.
6. Der Webserver liefert anschließend die Daten an den Client (Browser) zurück.

2.6.2. Beispiel mit VO



1. Zunächst startet ein Client (Browser) eine Anfrage für eine EXE-Datei. Dies geschieht über die Eingabe einer URL oder durch anklicken eines Links. Die Anfrage wird über das Internet an den entsprechenden Server weitergeleitet.
2. Der Server (Webserver) nimmt die Anfrage entgegen und startet das VO-Programm von seiner Festplatte.
3. Anschließend übergibt der Webserver den Anfrage-String an das VO-Programm.
4. Das VO-Programm arbeitet wie wir es gewohnt sind.
5. Anschließend gibt das VO-Programm das Ergebnis seiner Arbeit (meist in Form einer HTML-Datei, aber auch Bilder oder PDF-Dokumente sind möglich) an den Webserver zurück.
6. Der Webserver liefert anschließend die Daten an den Client (Browser) zurück.

2.7. Was wird benötigt, um einen Internet-Server mit VO einzurichten:

- Betriebssystem Windows 2000 Server oder 2003 Server (Testhalber geht auch Windows 2000 und Windows XP)
- Ein HTML-Server (IIS oder Apache, für Testbetrieb genügt auch der Jana-Server)
- Standleitung (symetrisch!), fixe IP-Adresse
- Hardware-Firewall, Virenscanner
- FTP-Zugriff oder Fernwartungssoftware (z.B. PC-Anywhere).
- Wartungstools (Serverüberwachung)

3. Programmierung mit VO

3.1. Unterschied zur herkömmlicher Programmierung

Wenn Serverseitig VO als Programmiersprache verwendet wird, so ergeben sich folgende Unterschiede:

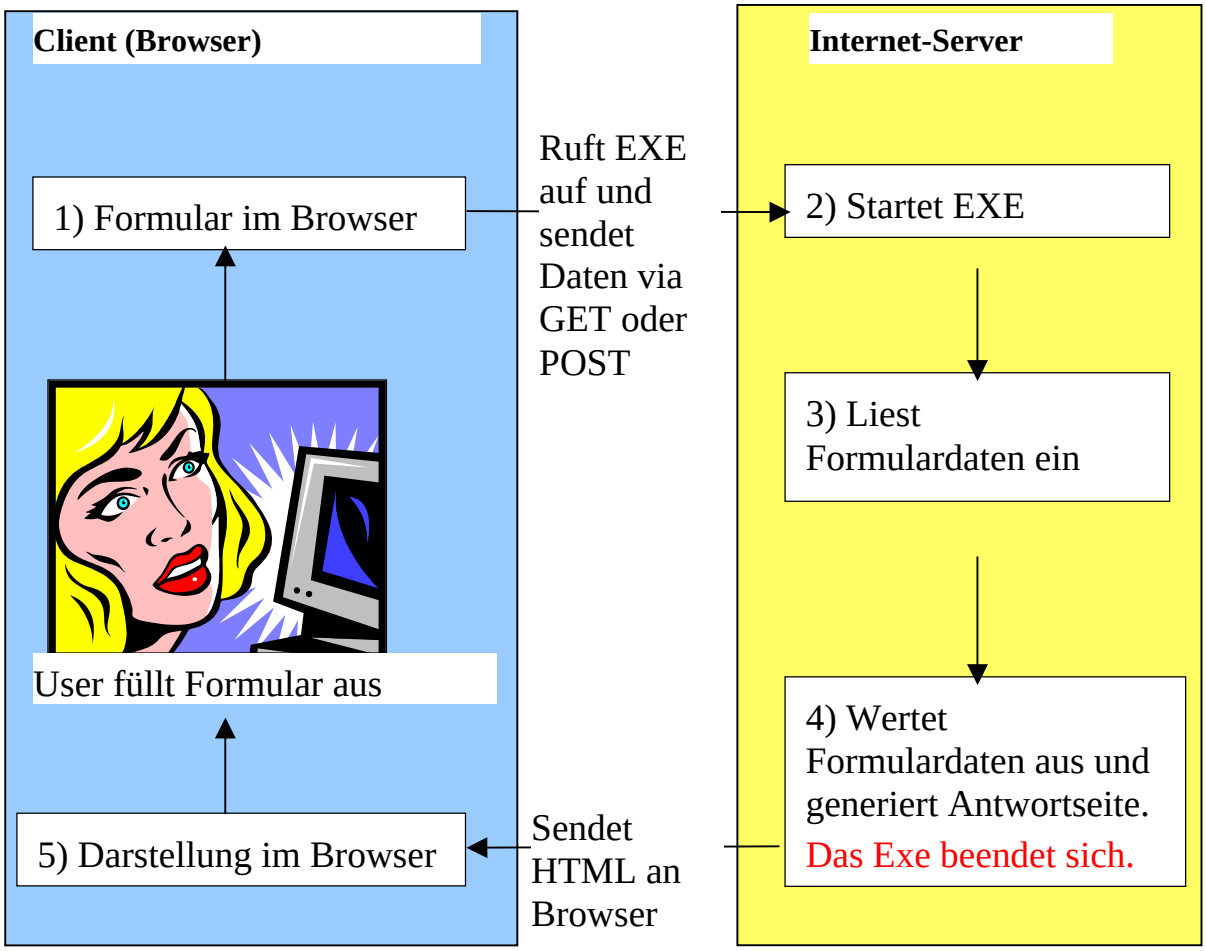
- Es besteht keine Verbindung zwischen der Applikation und dem Clienten
- Die GUI-Klasse von VO wird durch den Internet-Browser ersetzt

Alles andere bleibt bestehen (!)

3.2. Kommunikation Browser – VO-Serverprogramm

3.2.1. Ablauf der Kommunikationischen zwischen Client und Server

Die Basis einer Internetapplikation mit VO ist ejn Formular, geschrieben in HTML



1. Ein Formular wird im Browser dargestellt. Der User füllt es aus und drückt den „Submit“-Button
2. Auf dem Server wird das EXE gestartet. Dieses wird im Formular definiert (Es kann aber auch ein PHP oder Pearl-Interpreter gestartet werden!)
3. Das EXE liest die vom User ausgefüllten Formulardaten in ein Array ein
4. Je nach vom User gewählter Aktion wird eine neue HTML-Seite als String erzeugt und zum Browser geschickt. Das EXE wird anschliessend beendet
5. Der Browser stellt die HTML-Seite dar

3.3. Wie erkenne ich den User ?

Das Exe bekommt eine Seite zugeschickt und muss nun erkennen, welcher User diese Seite abgeschickt hat.

3.3.1. Möglichkeiten zur Identifikation des Absenders:

- IP-Adresse
  - Problem: Hinter einem Router haben alle User dieselbe IP-Adresse
- Cookie
  - Problem: Viele User haben Cookies abgeschaltet
- Session ID‘s
  - Session ID wird in einem unsichtbaren Formularfeld gespeichert
  - Das EXE speichert Session-ID und dazugehörende Daten vor seiner Beendigung auf dem Server
  - Browser sendet diese ID mit den GET oder POST-Daten wieder zurück
  - Das EXE sucht in einer Datenbank die Session-ID
  - Problem: Session-ID ist manipulierbar

3.3.2. Wie können Manipulationen verhindert werden?

Ein User könnte versuchen, durch Manipulation der Seite die Identität eines anderen Users anzunehmen. Dies kann wie folgt verhindert werden:

- Lange, komplizierte ID mit Verschlüsselung
- Gültigkeitsdauer der ID beschränken

Damit hat ein Manipulateur keine Chancen, innerhalb nützlicher Frist eine Session-ID zu fälschen, er müsste zudem eine neue, im aktuellen Zeitraum aktuelle ID eines anderen Users erzeugen.

4. Werkzeuge

4.1. Browser

Zur Zeit haben folgende Browser eine Bedeutung:

- Firefox: 1.5 und 2.0, Netscape 7.1
- Opera 8.0 und 9.0
- Microsoft Internet Explorer 5.5, 6.0 und 7.0
- Safari 1.2 bis 2.0 (Mac)
- Konqueror (Linux)

Kein einziger dieser Browser ist in der Lage, alle HTML-Befehle fehlerfrei auszuführen (!)

Am Besten funktioniert der Firefox, mit Abstand am schlechtesten der IE 5.5 und IE 6.0

Um eine Seite erstellen zu können, welche mit allen Browsern funktioniert, bleibt nichts anderes übrig, als die Seite auf jedem einzelnen Browser auszutesten.

In der Praxis genügt der Firefox 1.5, der IE 6,0 und ev. noch Opera 7

4.2. Add-Ons für Firefox

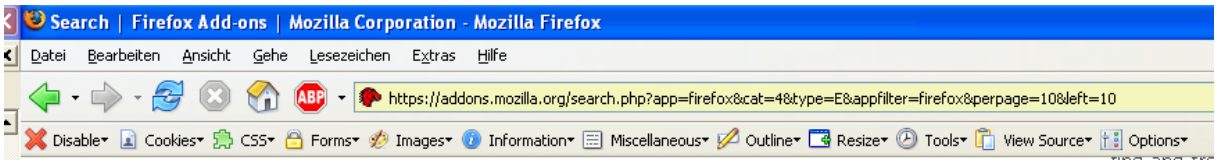
Für den Firefox gibt es einige sehr nützliche Erweiterungen:

4.2.1. IE Tab

Durch einen einfachen Klick auf ein Symbol lassen sich Seiten im Firefox mit der IE-Rendering-Maschine darstellen

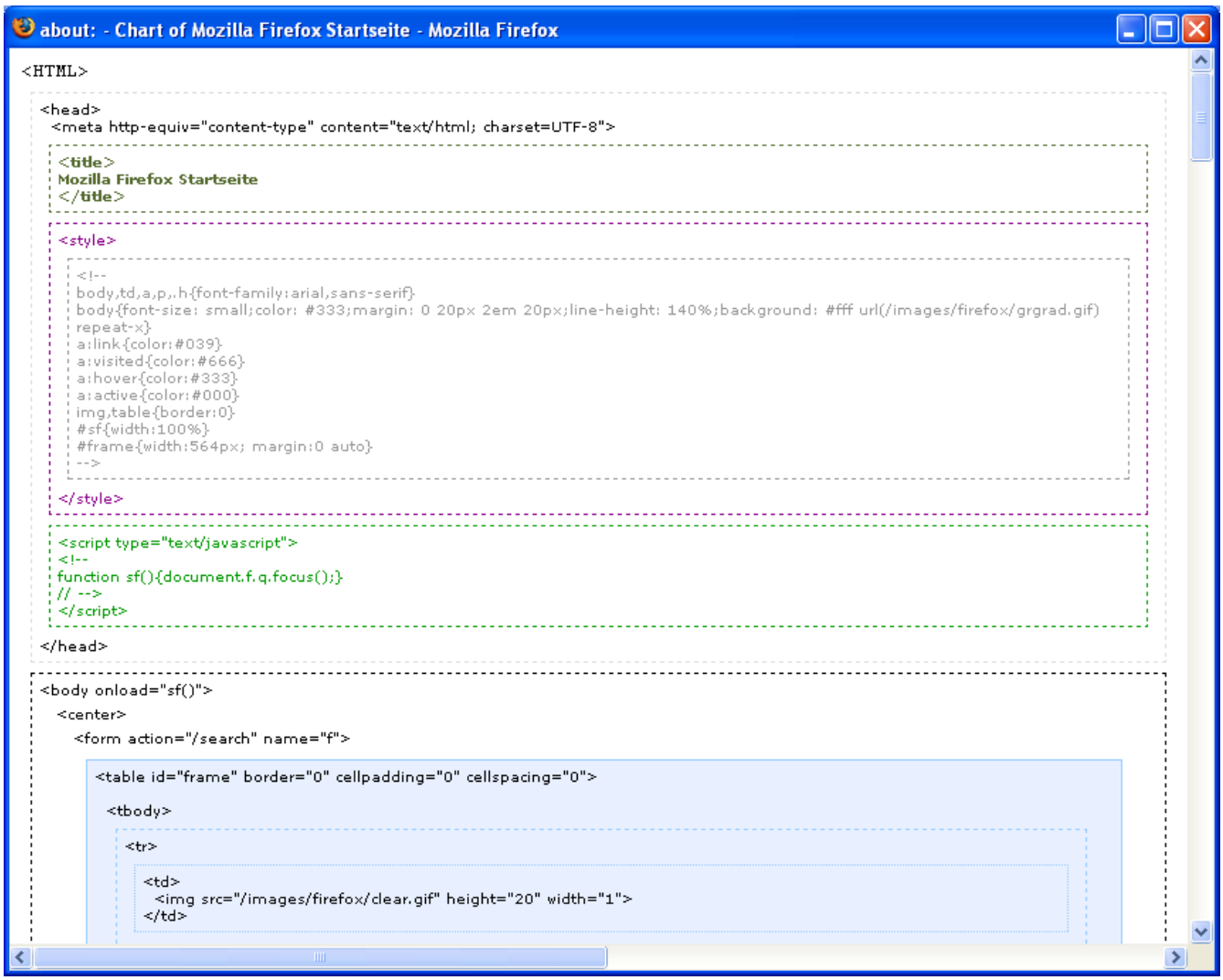
4.2.2. Web Developer

Toolbar mit extrem nützlichen Werkzeugen



4.2.3. View Source Chart

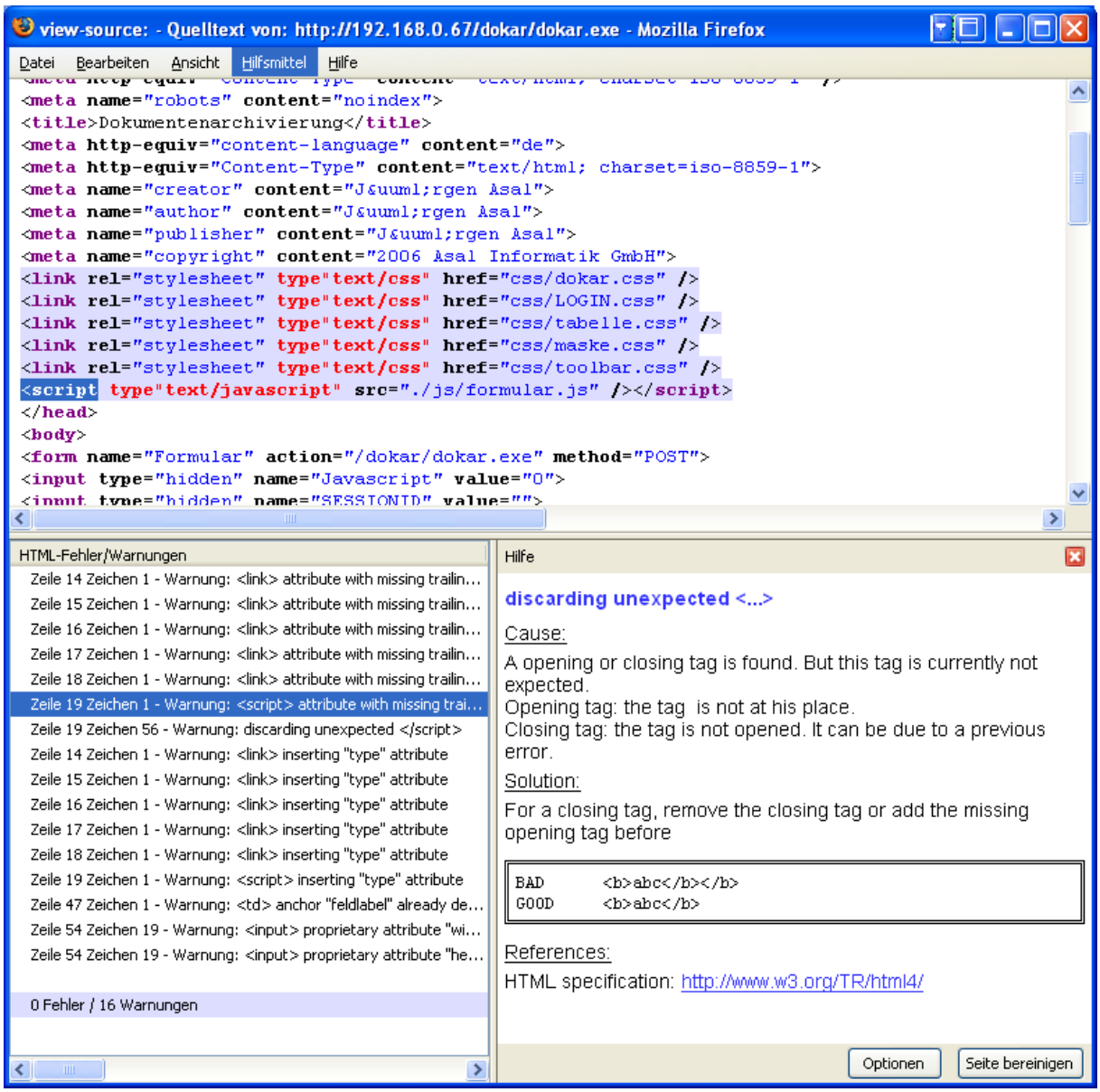
Zeigt den Quelltext formatiert an. Nicht geschlossene Tags lassen sich sehr einfach erkennen.



4.2.4. HTML-Validator

Zeigt Fehler in einer geladenen HTML-Site an.

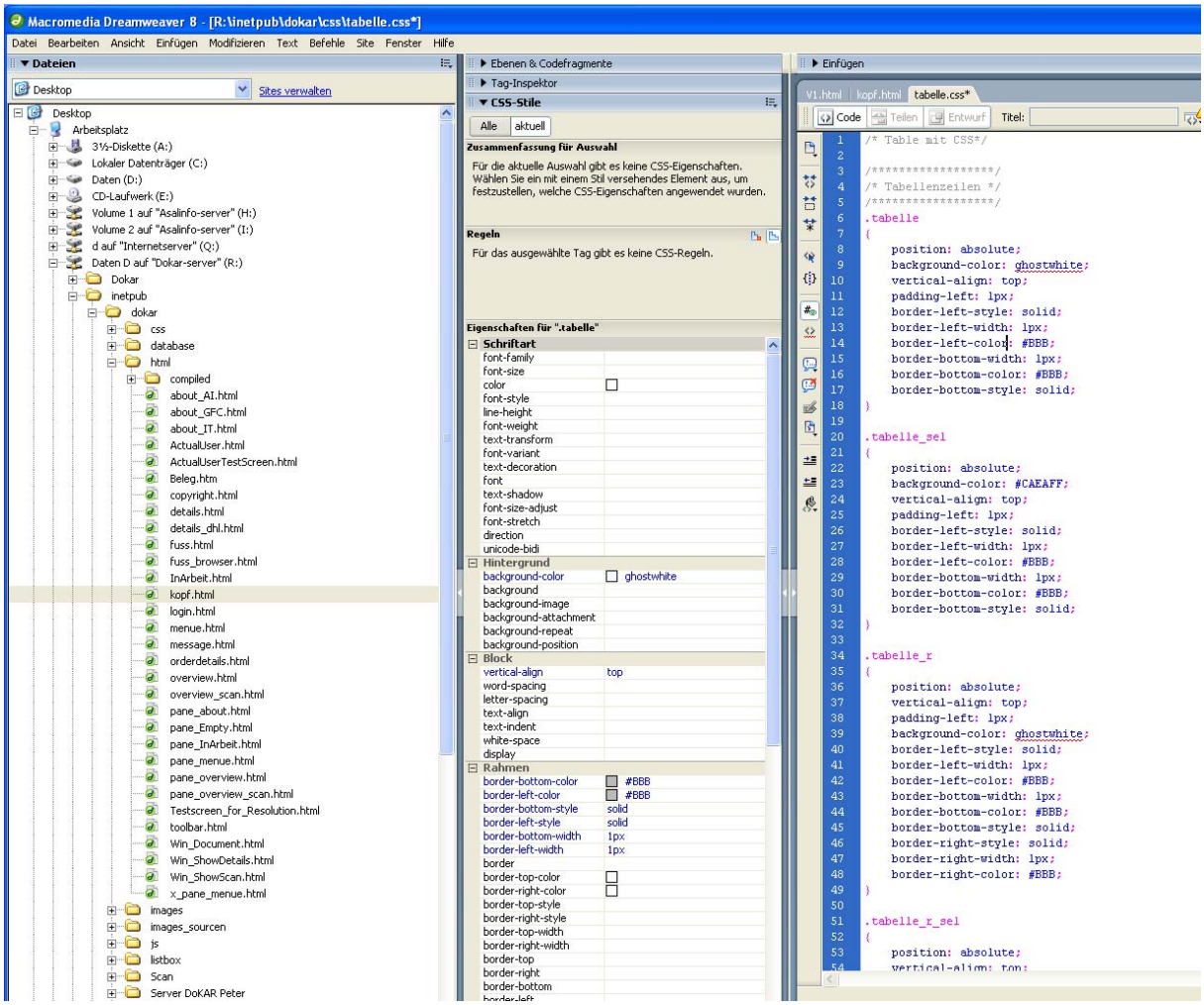
Da die Browser extrem Fehlertolerant sind, werden Fehler häufig nicht bemerkt!



4.3. HTML-Editoren

4.3.1. DreamWeaver

Bewährt hat sich der DreamWeaver von Marcomedia. Er bietet eine sehr gute Unterstützung bei CSS-Codes und hat einen excelenten Editor eingebaut. Zudem lässt er den Source-Code der HTML-Seite „in Ruhe“ und baut nicht unaufgefordert irgendwelchen Code ein.



4.3.2. Frontpage

Frontpage baut ständig eigenen Code in die Seite ein, zum Teil sogar Microsoft-proprietäre Elemente. Aus diesem Grund ist Frontpage *absolut* ungeeignet!

4.3.3. Super-HTML

Super-HTML ist ein simpler Freeware-Editor. Er bieten aber keinerlei Unterstützung bei CSS-Entwicklung an.

4.3.4. Adobe Go-Live

Mit Go-Live habe ich keine Erfahrung, er ist aber ähnlich wie Frontpage aufgebaut und hat dieselben Probleme

4.3.5. Word und Openoffice

Erzeugt aufgeblähten und kaum lesbaren Code

4.3.6. FireFox-Plugins

Es gibt auch gratis einige Plugins zu Firefox, um damit HTML-Seiten zu editieren. Vielleicht eine Alternative, wenn man nur etwas experimentieren möchte.

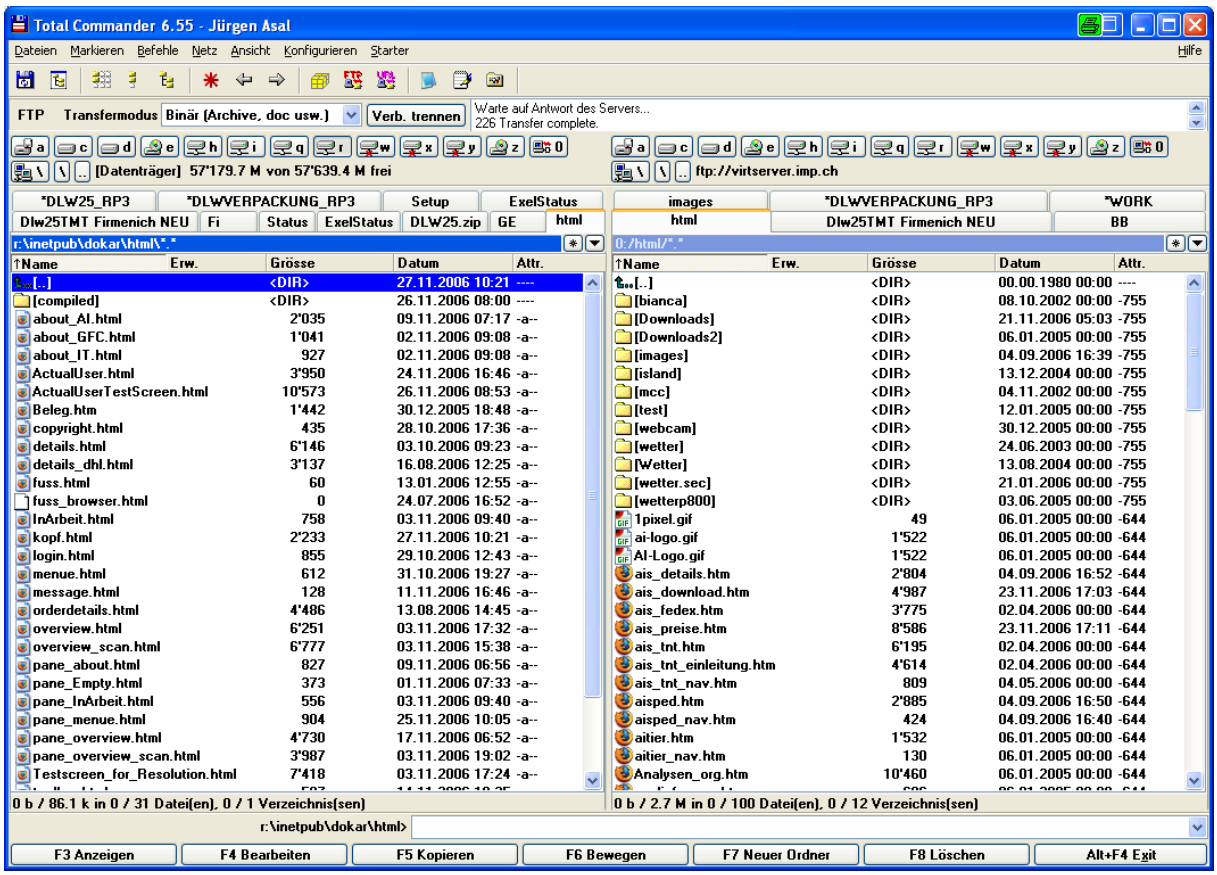
4.3.7. NotePad

Für echte Hardcore-Programmierer. Keinerlei Hilfen irgendwelcher Art, aber HTML ist ja nur eine Textdatei.

4.3.8. FTP-Transfer

Der „Total Commander“ bietet den absoluten Luxus zum Hochladen der eigenen Seiten. Links die lokale Harddisk, rechts der Internetserver.

http://www.ghisler.com/



5. Aufbau einer HTML-Seite

5.1. HTML-Versionen

Im Laufe der Zeit hat es verschiedene HTML-Versionen gegeben. Zur Zeit aktuell ist die Version 4.01. auf http://de.selfhtml.org/ findet man eine erstklassige Dokumentation.

5.2. Grundgerüst einer HTML-Seite

Beispiel: (V1)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>

<head>
<title>Dies ist der Dokumententitel. Er wird in der Titelzeile des Browsers
angezeigt</title>  </head>

<body>
<h1>Dies ist ein Titel</h1>
Dies ist Fliesstext
</body>

</html>
```

Jede HTML-Seite besteht aus mindestens den obigen Breichen:

- DOCTYPE-Zeile
- <HTML>-Tag
- <HEAD>-Tag
- <BODY>-Tag

5.2.1. Die DOCTYPE-Zeile

Hier wird definiert, nach welchem Standard sie Seite aufgebaut ist. Es gibt eine verwirrende Anzahl von Varianten. Für unsere Zwecke ist folgende Variante am sinnvollsten:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

Hier wird definiert, dass es sich um eine HTML-Seite mit der Version 4.01 handelt. (Möglich wären z.B. auch HTML 2.0, HTML 3.2), dass die BEFEHLE der Seite in Englisch geschrieben sind und dass die HTML-Konventionen strikt einzuhalten sind. Dieses STRICT ist der wichtigste Teil der Definition. Möglich wären noch „Transitional“ und „Frameset“.

Mit genau dieser Zeile wird die bestmögliche Kompatibilität erreicht.

5.2.2. Der <HTML>-Tag

Dieser Tag sagt nicht weiter aus als dass nun HTML-Code folgt. Wichtig: Den schliessenden Tag </HTML> am Ende der Seite nicht vergessen!

5.2.3. Der <HEAD>-Tag

Hier wird der Titel des Dokumentes sowie gewisse Grundeinstellungen wie Hintergrundfarbe und Farbe der Links definiert.

Jede HTML-Datei **muss** einen Titel erhalten. Das ist aus folgenden Gründen besonders wichtig:

- Der Titel der Datei wird bei der Anzeige im Web-Browser in der Titelzeile des Anzeigefensters angezeigt.
- Der Titel der Datei wird bei der Anzeige im Web-Browser in Karteireitern (Tabs) angezeigt.

- Der Titel der Datei wird vom Web-Browser beim Setzen von Lesezeichen (Bookmarks, Favoriten) auf die Datei verwendet.
- Der Titel der Datei wird im Web-Browser in der Liste der bereits besuchten Seiten angezeigt.
- Der Titel der Datei dient im Web vielen automatischen Suchprogrammen als wichtiger Input. Wenn die Datei zu den Suchtreffern einer Suche gehört, bieten viele Suchmaschinen den Titel der Datei als Anklickbahren Verweis an.

Ebenfalls lassen sich sogenannte META-Angaben definieren: In Meta-Angaben können Sie verschiedene nützliche Anweisungen für Web-Server, Web-Browser und automatische Suchprogramme im Internet ("Robots") notieren. Meta-Angaben können Angaben zum Autor und zum Inhalt der Datei enthalten. Sie können aber auch HTTP-Befehle absetzen, zum Beispiel zum automatischen Weiterleiten des Web-Browsers zu einer anderen Adresse.

5.2.4. Der <BODY>-Tag

Hier steht das eigentliche HTML-Dokument. Auch hier am Ende nicht das </BODY> vergessen

5.3. Absätze

Der Text im Body-Breich muss in Absätze unterteilt werden. Text einfach so hinzuschreiben ist nicht mehr zulässig.

Ein Absatz beginnt mit und endet mit

Es gibt noch weitere Absatz-Tags wie <h1> </h1> für Überschriften. Ein Mischen der Tags innerhalb eines Absatzes ist nicht erlaubt, wird aber von allen Browsern toleriert (z.B. <h1> TEXT </h1>, ist völliger Unsinn).

Beispiel:

```
<body>
<h1>Dies ist eine Hauptüberschrift</h1>
<h2>Dies ist eine untergeordnete Überschrift</h2>
<p>Dies ist ein Absatz. Dies ist Fliesstext. Und hier noch etwas Text.<p>
<p>Dies ist ein weiterer Absatz. Dies ist Fliesstext. Und hier noch etwas Text.<p>
</body>
```

Die Zeilenschaltung hat keine Bedeutung. Mann kann genauso wie folgt schreiben:

```
<body><h1>Dies ist eine Hauptüberschrift</h1><h2>Dies ist eine untergeordnete
Überschrift</h2><p>Dies ist ein Absatz. Dies ist Fliesstext. Und hier noch etwas
Text.<p><p>Dies ist ein weiterer Absatz. Dies ist Fliesstext. Und hier noch etwas
Text.<p></body>
```

Eine Zeilenschaltung, welche NICHT zu einem neuen Absatz führen soll, wird als <br /> eingegeben.
 ist die Kurzform vom
</br> (Tags immer schliessen!)

5.4. Umlaute

Alle Umlaute sollten / müssen als Code eingegeben werden. Ein Ü ist z.B. Ü

Was passiert, wenn man es nicht so macht, konnte lange Zeit auf Oscars Seite bewundert werden.

5.5. Formatierung

Es gibt zwei Möglichkeiten, Formatierungen durchzuführen:

5.5.1. Formatierung im HTML-Code

Beispiel einer Formatierung im HTML-Code:

```
<p align="right">Dies ist ein Absatz, der rechtsb&uuml;ndig ausgerichtet ist.</p>
<p><font color="#FF0000">Knallroter Text</font></p>
```

Der grosse Nachteil ist die Vermischung zwischen Formatierung und Inhalt

5.5.2. Formatierung mit CSS

Mit CSS ist die Formatierung getrennt vom Inhalt. Man kann dies am ehesten mit den Formatvorlagen im Word vergleichen. Der Text steht in der DOC-Datei, die Formatierungen in der DBT-Datei

Beispiel:

Inhalt einer HTML-Datei

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Vortrag Konstanz</title>
</head>

<body>
<h1>Dies ist eine &Uuml;berschrift</h1>
<p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und
fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst
immer noch, der Fliesstext.</p>
<p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich hin fliesst.
Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er
Fliesstext. Er fliesst immer noch, der Fliesstext. So, jetzt reicht es.</p>
</body>
</html>
```

Ergebnis im Browser:

Dies ist eine Überschrift

Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst immer noch, der Fliesstext.

Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst immer noch, der Fliesstext. So, jetzt reicht es.

Beispiel einer Formatierung mit CSS:

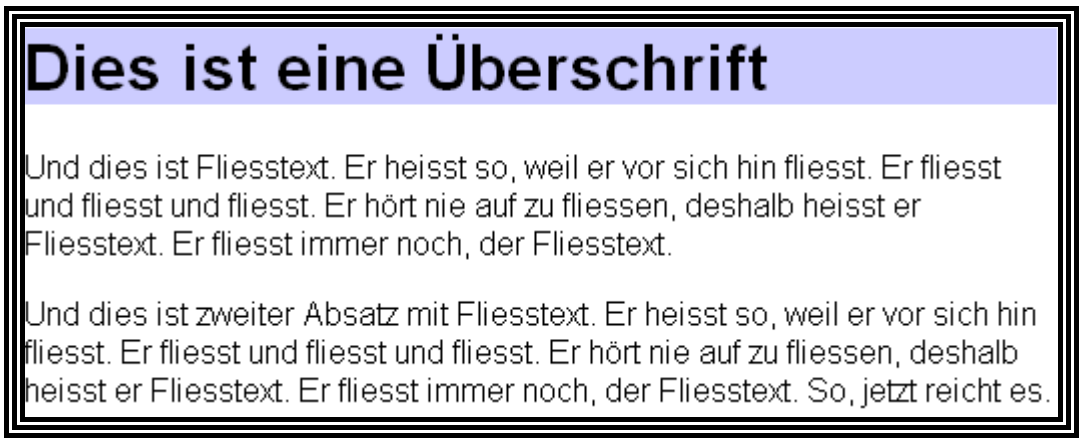
Inhalt der HTML-Datei

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Vortrag Konstanz</title>
<link rel="stylesheet" type="text/css" href="Vortrag2.css">
</head>

<body>
<h1>Dies ist eine &Uuml;berschrift</h1>
<p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und
fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst
immer noch, der Fliesstext.</p>
<p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich hin fliesst.
Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er
Fliesstext. Er fliesst immer noch, der Fliesstext. So, jetzt reicht es.</p>
</body>
</html>
```

Inhalt der CSS-Datei „Vortrag2.css“

```
body {
    font-family: Arial, Helvetica, sans-serif;
}

h1 {
    background-color: #CCCCFF;
}
```

Ergebnis im Browser:



Wir haben jetzt den Inhalt (*.HTML) getrennt von der Formatierung (*.CSS)

Wir haben jetzt den <BODY> und den <H1>-Tag umformatiert. Es lassen sich aber auch eigene Tag’s, sog. Klassen, definieren.

Beispiel:

```
:
:

<body>
<h1>Dies ist eine &Uuml;berschrift</h1>
<p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und
fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst
immer noch, der Fliesstext.</p>
<p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich hin fliesst.
Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er
Fliesstext. Er fliesst immer noch, der Fliesstext. So, jetzt reicht es.</p>
<h1>Dies ist noch eine &Uuml;berschrift</h1>
<p class="hervorgehoben">Und dies ist Fliesstext. Er heisst so, weil er vor sich hin
fliesst. Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er
Fliesstext. Er fliesst immer noch, der Fliesstext.</p>
</body>
</html>
```

Inhalt der CSS-Datei „Vortrag3.css“

```
body {
    font-family: Arial, Helvetica, sans-serif;
}

h1 {
    background-color: #CCCCFF;
}

.hervorgehoben{
    color: #999999;
    font-weight: bold;
    font-size: 14px;
}
```

Wichtig ist der Punkt vor “hervorgehoben”. Er bezeichnet eine selbstdefinierte Klasse.

Und das sieht so aus:

Der neue Absatz ist jetzt hellgrau und fett. Soweit logisch. Aber wieso ist die Schrift jetzt immer noch Arial, obwohl gar keine Schriftart definiert wurde und die Default-Schrift „Times“ ist?

5.5.3. Klassen und Vererbung

Klassen können, wie in VO, Eigenschaften vererben.

Wir haben definiert, dass der <BODY>-Tag mit Arial formatiert ist.

Sowohl <H1> als auch <P> und „hervorgehoben“ sind in der Hirarchie unterhalb des Body-Tags und erben deshalb seine Eigenschaften. Es gibt kein explizites „Inherit“, die Vererbung wird nur durch die Anordnung der Tags im HTML-Text bestimmt!

Unser Beispiel nach Ebenen angeordnet sieht so aus:

```
<HTML>
        <HEAD>
        </HEAD>
        <BODY>
                <H1>
                        (Text)
                </H1>
                <P>
                        (Text)
                </P>
                <P>
                        (Text)
                </P>
                <H1>
                        (Text)
                </H1>
                <P Class=“hervorgehoben”
                        (Text)
                </P>
        </BODY>
<HTML>
```

Alles, was unterhalb der Ebene <BODY> ist, erbt die Schriftart ARIAL, also <H1>, <P> und die Klasse „hervorgehoben“

Damit die Hierarchie immer ersichtlich ist, empfehle ich **dringend**, den HTML-Code einzurücken (wie bei VO <g>)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/x
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        <title>Vortrag Konstanz</title>
        <link rel="stylesheet" type="text/css" href="Vortrag3.css">
    </head>

    <body>
        <h1>Dies ist eine &Uuml;berschrift</h1>
        <p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fli
        <p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich
        <h1>Dies ist noch eine &Uuml;berschrift</h1>
        <p class="hervorgehoben">Und dies ist Fliesstext. Er heisst so, weil er vor si
        <p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich
        <h2>Dies ist eine Unter-&Uuml;berschrift</h2>
        <p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fli
        <p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich
    </body>
</html>
```

Leider muss ich jetzt die schöne Welt der Vererbung wieder zerstören:

- Es werden nicht alle Eigenschaften vererbt.
- Die Vererbung funktioniert nicht immer korrekt
- Nicht alle Eigenschaften werden korrekt ausgeführt

Wie findet man heraus, welche Eigenschaft nun vererbt wird resp. welche Vererbungen funktionieren?
Ich kenne nur eine Methode:

- Try-and-error

Speziell beim IE funktioniert vieles nicht, beim FireFox funktioniert aber das Meiste.

5.6. CSS-Versionen

Währen man bei HTML immer von der Version 4.01 ausgehen kann, gibt es bei CSS mehrere Versionen, welche sich im Befehlssatz unterscheiden:

CSS 1.0

CSS 2.0

CSS 2.1

CSS 3.0

Leider unterstützt kein einziger Browser alle Befehle der Version 1.0, geschweige denn die Befehle der höheren Versionen. Die Version 2.0 wird von den Browsern einigermassen unterstützt, so dass man deren Befehle mit guten Erfolgschancen verwenden kann. Die Version 3.0 wird nahezu gar nicht unterstützt. Wenn ein Befehl der Version 3.0 funktioniert, dann ist dies ein riesen Glück.

In der Literatur gibt es genügend Tabellen, welcher Browser welche Befehle unterstützt.

5.7. <DIV>

Das -Tag ist einer der wichtigsten und mächtigsten Befehle überhaupt.

Mit dem lässt sich ein Dokument in Abschnitte unterteilen. Der <p>-Tag tut etwas ähnliches, er unterteilt einen Text in Abschnitte (und fügt eine Zeilenschaltung ein).

Der ist aber (zuerst) unsichtbar. Erst wenn der mit einer CSS-Klasse formatiert wird, tut sich etwas.

Beispiel:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        <title>Vortrag Konstanz</title>
        <link rel="stylesheet" type="text/css" href="Vortrag4.css">
    </head>

    <body>
        <div>
            <h1>Dies ist eine &Uuml;berschrift</h1>
            <p>Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. E
            <p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor
        </div>

        <div>
            <h1>Dies ist noch eine &Uuml;berschrift</h1>
            <p class="hervorgehoben">Und dies ist Fliesstext. Er heisst so, weil er v
            <p>Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor
        </div>
    </body>
</html>
```

Auf dem Bildschirm tut sich noch überhaupt nichts!

In der CSS definieren wir nun:

```
.box_links{
     position: absolute;
     width: 200px;
}

.box_rechts{
     position: absolute;
     width: 200px;
     left: 220px;
}
```

Den beiden -Tags weisen wir nun eine Klasse zu

```
 9  <body>
10          <div class="box_links">
11              <hl>Dies ist eine &Uuml;berschrift</hl>
12              <p>Und dies ist Fliesstext. Er heisst so, weil e
13              <p>Und dies ist zweiter Absatz mit Fliesstext. E
14          </div>
15
16      <div class="box_rechts">
17          <hl>Dies ist eine &Uuml;berschrift</hl>
18          <p class="hervorgehoben">Und dies ist Fliesstext. Er
19          <p>Und dies ist zweiter Absatz mit Fliesstext. Er he
20      </div>
21  </body>
22  </html>
```

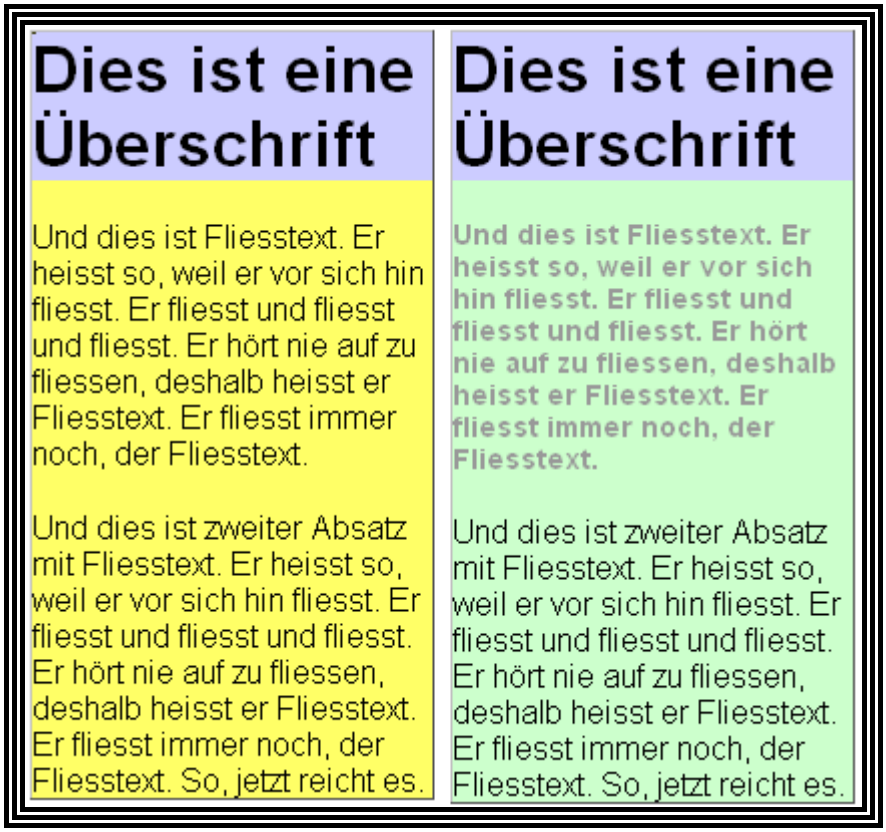
Der Bildschirm sieht nun so aus:

Wir haben nun zwei so genannte Layers oder Ebenen definiert. Diese kann man sich wie Klarsichtfolien vorstellen, welche auf dem Untergrund liegen und mit Positions- und Grössenangaben platziert und Zugeschnitten werden können.

Zur Verdeutlichung färben wir diese beiden Folien ein:

```
.box_links{
	position: absolute;
	width: 200px;
	background-color: #FFFF66;
}

.box_rechts{
	position: absolute;
	width: 200px;
	left: 220px;
	background-color: #CCFFCC;
}
```

Nun sieht es so aus:

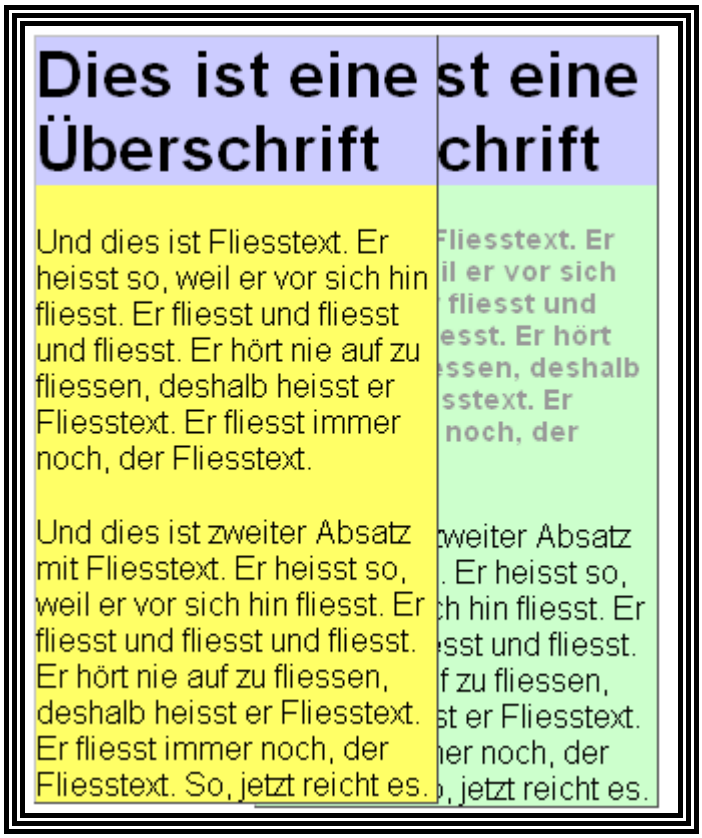


Die Überschriften haben die Hintergrundfarbe behalten, sie haben eine höhere Priorität, weil sie auf der Folie quasi draufliegen.

Natürlich können sich die Folien auch überlappen:

```
.box_links{
	position: absolute;
	width: 200px;
	background-color: #FFFF66;
	z-index: 2;
}

.box_rechts{
	position: absolute;
	width: 200px;
	left: 120px;
	background-color: #CCFFCC;
	z-index: 1;
}
```

Der Z-Index bestimmt dabei die Reihenfolge der Folien.

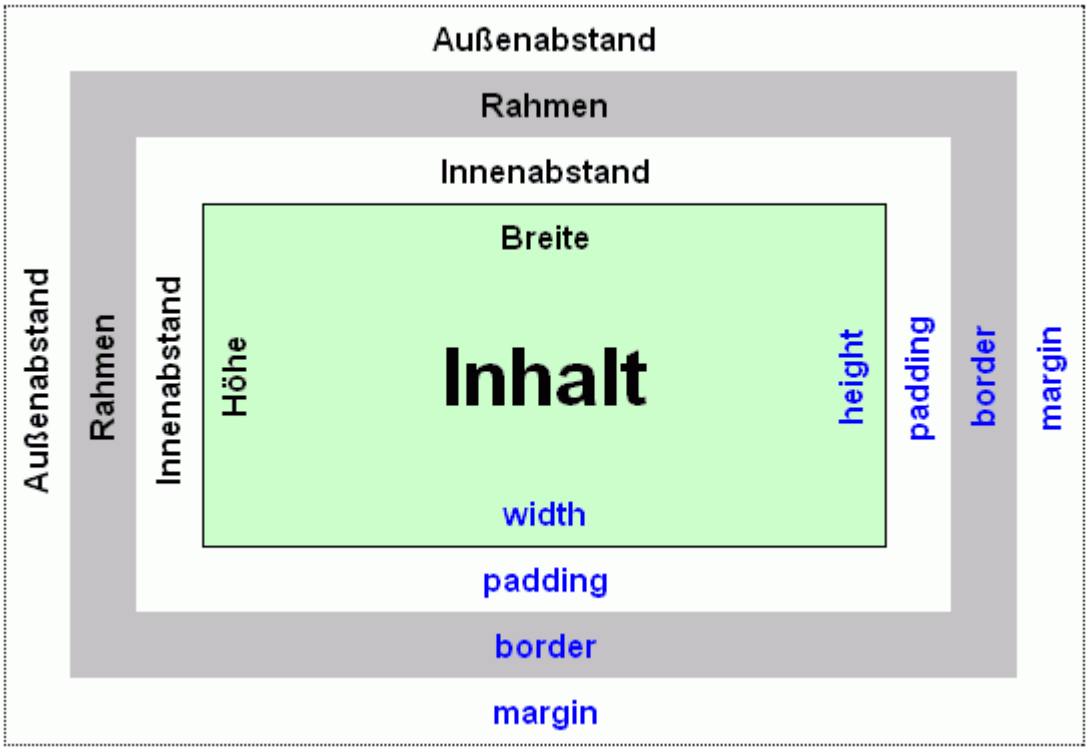

Dies ist eine Überschrift
Und dies ist Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst immer noch, der Fliesstext.
Und dies ist zweiter Absatz mit Fliesstext. Er heisst so, weil er vor sich hin fliesst. Er fliesst und fliesst und fliesst. Er hört nie auf zu fliessen, deshalb heisst er Fliesstext. Er fliesst immer noch, der Fliesstext. So, jetzt reicht es.

5.8. Das BOX-Modell

Wer jetzt nicht zusammengezuckt ist, hat noch nie HTML programmiert <g>.

Das BOX-Modell ist einer der schlimmsten Übel des Internet.

Darunter versteht man die Formatierung einer Box. Eine Box ist z.B. ein umrandeter Text

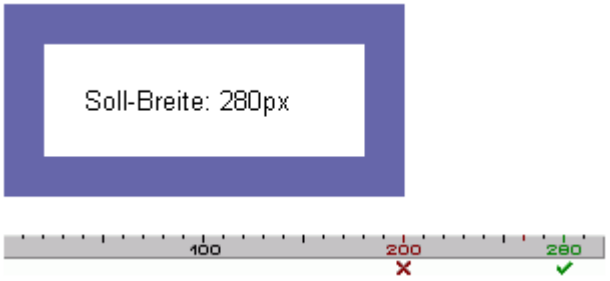


Beispiel:

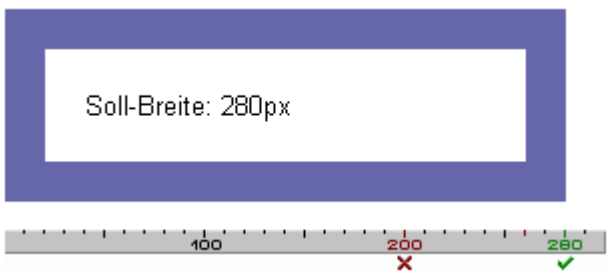
Wird ein Element mit einer Breite von 200 Pixel, einer Höhe von 100 Pixel und einem Innenabstand und Rahmen von je 20 Pixel Stärke auf allen Seiten definiert, beträgt die tatsächliche Breite letztendlich 280 Pixel (20 für border-left, 20 für padding-left, 200 für width, 20 für padding-right und 20 für border-right), die Höhe 180 Pixel (20 für border-top, 20 für padding-top, 100 für height, 20 für padding-bottom und 20 für border-bottom). Ein zusätzlich definierter Außenabstand müsste zu diesen Werten nochmals addiert werden.

Der Box-Modell-Fehler des Internet Explorers

Als "Box Model Bug" wird der Fehler in älteren Windows-Versionen des Internet Explorers (einschließlich 5.5) bezeichnet, die Innenabstände und Rahmenstärken entgegen der Spezifikation nicht zur Gesamtbreite zu addieren - dies ist nur beim Außenabstand korrekt der Fall.



Internet Explorer



Firefox

6. Interaktion mit dem Webserver

Damit der User eine Interaktion mit dem Internetserver auslösen kann, gibt es nur zwei Möglichkeiten:

Er klickt auf einen LINK oder er drückt den SUBMIT-Button eines HTML-Formulars.

Eine andere (zuverlässige) Methode gibt es nicht!

6.1. Links

6.1.1. Grundaufbau eines Links

Ein Link sieht so aus: www.asal-info.ch

Er ist defaultmässig blau und unterstrichen.

Zum Glück lässt sich so ein Link praktisch nach Belieben formatieren.

```
  <a href="http://www.heise.de/newsticker/">Heise Newsticker</a>
```

Ein Link beginnt mit dem Tag <a> und enthält immer ein Verweisziel, welches aufgerufen wird, wenn der User den Link anklickt, sowie einen Text, welcher blau unterstrichen angezeigt wird.

Wenn man so einen Link anklickt, wird der Inhalt des href-Attributes in die ULR-Zeile des Browsers kopiert und ein <RETURN> ausgelöst.

Diese ULR wird an den Web-Server geschickt und dort bearbeitet.

6.1.2. Link mit Parameter

Mit diesem simplen Link können wir aber nicht viel anfangen. Wir können aber noch Zuatzinformationen als Parameter mitgeben:

```
<a href="/dokar/dokar.exe?sessionid=152299585200611291903 26&amp;pane_id=USER_PANE_01_01">
</a>
```

Analysieren wir diese Zeile:

```
<a href="/dokar/dokar.exe?sessionid=152299585200611291903 26&amp;pane_id=USER_PANE_01_01">
```

<a	Einleitung des Links
href="/dokar/dokar.exe	Ruft auf dem Server DOKAR.EXE auf
?	Kennzeichen, dass noch Parameter folgen
sessionid=152299585200611291903 26	1. Parameter
&	Trennzeichen zwischen Parametern (&)
pane_id=USER_PANE_01_01	2. Parameter
">	Ende des Links

Dies führt zum Start des VO-Programmes „DOKAR.EXE“. Das Exe kann sich nun die Parameter mittels der GET-Methode einlesen. (Siehe Norberts Vortrag „GET or POST“). Dafür gibt es zum Glück eine VO-Klasse.

6.1.3. Formatierung eines Links

Diese hübsche Schaltfläche ist auch ein Link, auch wenn er gar nicht so aussieht

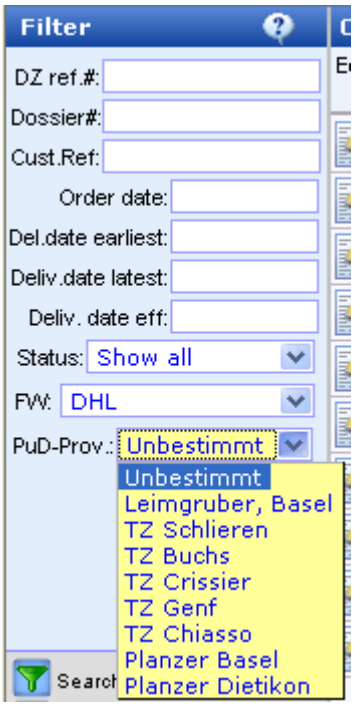

```
<a href="/dokar/dokar.exe?sessionid=1522995852006112919032 6&amp;pane_id=USER_PANE_01_01">
<div class="pane_menue_selected" style="top:20px; height:19px">
<div class="pane_menue_text">My Account</div>
<img class="pane_menue_icon" src="./images/FOLDERUNCOMPRESSED.GIF" style="top:2px;" />
</div>
</a>
```

```
/**********************/
/* Pane-Menue-Bereich */
/**********************/
.pane_menue, .pane_menue_selected, .pane_menue_text {
     position: absolute;
     font-family: Verdana, Arial, Helvetica, sans-serif;
     font-size: 10px;
     text-align: left;
     border: 1px solid #002D96;
     width: 100%;
}

.pane_menue_text {
     position: absolute;
     border: 0px;
     padding-top: 3px;
     left: 25px;
}

.pane_menue {
     background-image: url(../images/OutlookButtonBlau.gif);
     background-color: #8CB2E7;
}

.pane_menue_selected {
     background-image: url(../images/OutlookButtonBlau.gif);
     background-color: #8CB2E7;
     font-weight: bold;
}

.pane_menue_icon {
     position: absolute;
     height: 16px;
     width: 16px;
     left: 5px;
     border: 0px;
 }

a:hover .pane_menue{
     background-image: url(../images/1px.gif);
     background-color: #CAEAFF;
     color: #00F;
}

a:hover .pane_menue_selected{
     background-image: url(../images/1px.gif);
     background-color: #CAEAFF;
     color: #00F;
}
```

6.2. HTML-Formular

Wenn man ein HTML-Formular verwendet, so hat man einen Submit-Button zur Verfügung. Sobald dieser gedrückt wird, wird wiederum das EXE aufgerufen. Zudem werden sämtliche Formularfelder übergeben. Das EXE kann sich nun die Felder mit der POST-Methode einlesen. Die Parameterübergabe mittels GET wäre auch möglich, ist aber absolut nicht zu empfehlen, weil der User dann die Parameter in der ULR-Zeile sieht und manipulieren kann und weil die Länge der ULR-Zeile begrenz ist.

Beispiel eines Formulars:



6.2.1. Aufbau eines HTML-Formulars

Ein HTML-Formular wird immer vom <form>-Tag umrahmt. Hier wird definiert, wohin das Formular gesendet werden soll (an ein Programm namens „DOKAR.EXE“). Die Formulardaten sollen mit POST gesendet werden.

```
<form action="dokar/dokar.exe" method="POST">
   Formularfelder…..

   Formularfelder…..

   Formularfelder…..
   Sendebutton
</form>
```

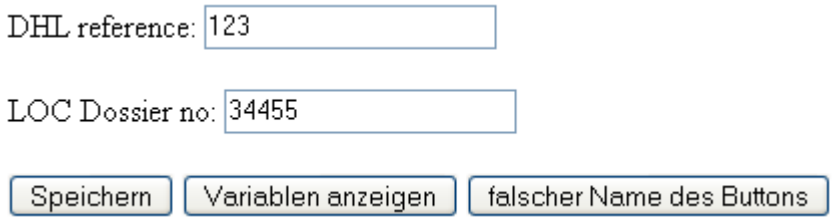
Zu Beachten:

- Das Formular wird erst übertragen, wenn der Sendebutton gedrückt wird!
- Es gibt keinerlei Möglichkeit, die Eingabe zu überprüfen, bevor Senden gedrückt wird! (Ausser über Javascript)
- Es gibt keine Datentypen sondern nur Textfelder!
- Es gibt keinerlei „Notify“-Events (z.B. ListboxSelect) etc., ausser man verwendet JavaScript

6.2.2. Beispiel eines einfachen Formulares

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
     <head>
          <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
          <title>Formular</title>
     </head>

     <body>
          <form name="Formular" action="/dokar/konstanz.exe" method="POST">
               <label>DHL reference:</label>
               <input name="sleBELEGNR" type="TEXT" value="" maxlength="35" />
               <br /><br />
               <label>LOC Dossier no:</label>
               <input name="sleINT_BELEG" type="TEXT" value="0" maxlength="35" />
               <br /><br />
               <input name="PB_SEARCH" type="submit" value="Speichern">
               <input name="PB_VARIABLEN" type="submit" value="Variablen anzeigen">
               <input name="PB_VARIABLE" type="submit" value="falscher Name des Buttons">
          </form>
     </body>
</html>
```

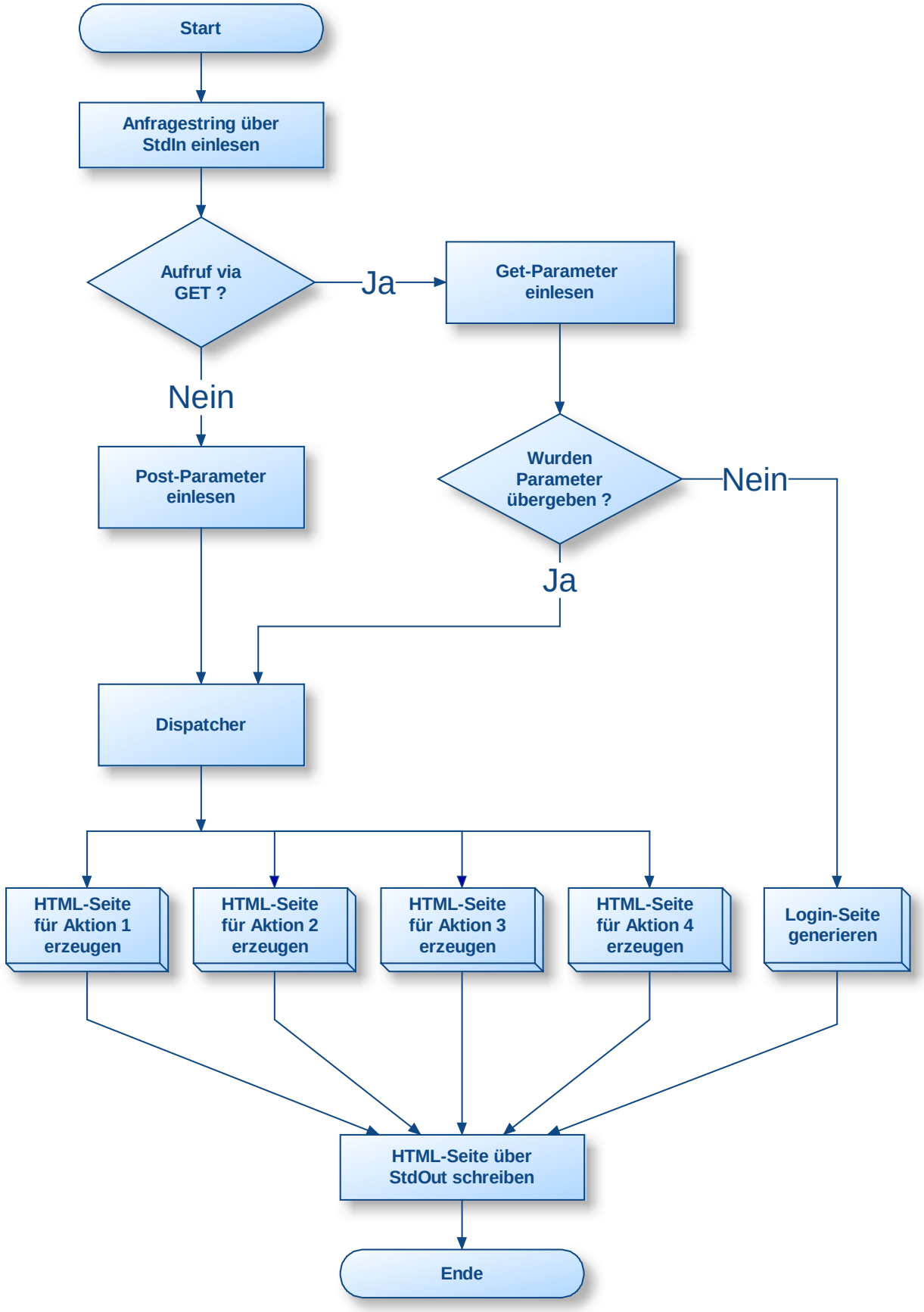
Anzeige im Browser



Anzeige im Debugger

```
===============
HTTP_USER_AGENT: Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.0.8) Gecko/20061025
Firefox/1.5.0.8
ENGINE          : GECKO
USER_AGENT      : FIREFOX
QUERY_STRING    :
Remote IP       : 192.168.0.65
Javascript      :
===============
Parameters    : 3
-------------------------------------------------
          SLEBELEGNR = 4711
         SLEINT_BELEG = 42
  OVERVIEW_PB_SEARCH = Anfrage abschicken
-------------------------------------------------
Method: POST
```

7. Grundaufbau des EXE‘s



7.1. Programmbeispiel

7.1.1. Funktion Start

```
FUNCTION Start

    LOCAL oCGI      AS StdHTTPContext
    LOCAL cHtml     AS STRING
    LOCAL oOldError AS USUAL
    LOCAL oError    AS USUAL
    LOCAL oLogFile  AS LogFile

    SetAnsi          (TRUE)
    SetExclusive     (FALSE)
    SetDeleted       (TRUE)
    SetCollation     (#CLIPPER)
    SetInternational (#CLIPPER)

    // Abschließend (!) Deutsches Datum und 1998 etc.
    // Nach Set Ansi On!
    SetDateCountry   (German)
    SetCentury       (TRUE)

    // Einstellungen für Dezimal-Komma und Tausender-Punkt
    SetDecimalSep    (Asc ("."))
    SetThousandSep   (Asc (","))

    IF File (WorkDir()+"debug")
     FErase(WorkDir()+"dokar.log")
    ENDIF

    oOldError := ErrorBlock( {|oErr| __MyError(oErr)} )

    BEGIN SEQUENCE
        oCGI     := StdHTTPContext{}
        cHtml    := oCGI:HTTPResponse()

        IF SLen(cHtml) > 0
            oCGI:Write(cHtml)
        ENDIF

    RECOVER USING oError
        oLogFile := LogFile{"RunTimeError",,.F.}
        cHtml := oCGI:HTTPErrorMessage(oError)
        oCGI:Write(cHtml)
        oLogFile:DumpError(oError)
    END SEQUENCE

    ErrorBlock( oOldError )

    oCGI:Close()

    RETURN
```

Interessant ist dieser Ausschnitt:

```
        oCGI     := StdHTTPContext{}
        cHtml    := oCGI:HTTPResponse()

        IF SLen(cHtml) > 0
            oCGI:Write(cHtml)
        ENDIF
```

7.1.2. Klasse StdHTTPContext

```
CLASS StdHTTPContext      INHERIT   A_HttpCgiContext

    PROTECT lDebug              AS LOGIC

    DECLARE METHOD Dispatch
```

7.1.3. Die INIT-Methode

In der Init - Methode wird der Anfragestring des Browsers analysiert und in eine Debug-Datei ausgegeben:

```
METHOD Init(cLogFile, lAppend) CLASS StdHTTPContext
    // Lets spy out arguments (get), data (post) and cookies

    LOCAL i          AS INT
    LOCAL nLen       AS INT
    LOCAL oLogFile   AS LogFile

    Default(@lAppend, .T.)

    SUPER:Init()

    lDebug := File("DEBUG")

    IF lDebug

        oLogFile := LogFile{cLogFile,.F.,lAppend}   // Logdatei anlegen

        oLogFile:DebugMsg( "===============")
        oLogFile:DebugMsg( "HTTP_USER_AGENT: " + SELF:GetServerVariable("HTTP_USER_AGENT") )
        oLogFile:DebugMsg( "ENGINE         : " + Symbol2String(SELF:_Engine))
        oLogFile:DebugMsg( "USER_AGENT     : " + Symbol2String(SELF:_UserAgent ))
        oLogFile:DebugMsg( "QUERY_STRING   : " + SELF:GetServerVariable("QUERY_STRING") )
        oLogFile:DebugMsg( "Remote IP      : " + SELF:_Remote_IP)
        oLogFile:DebugMsg( "Javascript     : " + SELF:GetParamValue("JAVASCRIPT"))
        oLogFile:DebugMsg( "===============")

        nLen := ALen(SELF:aArgs)
        oLogFile:DebugMsg( "Parameters    : "+AllTrim(Str(nLen)))
        oLogFile:DebugMsg( Replicate("-",50))

        // Alle Formularfelder ausgeben
        FOR i := 1 TO nLen
            IF ALen( SELF:aArgs[i] ) >=2
                oLogFile:DebugMsg(PadL(SELF:aArgs[i][1],20)+" = "+SELF:aArgs[i][2])
            ELSE
                oLogFile:DebugMsg(SELF:aArgs[i][1])
            ENDIF
        NEXT

        oLogFile:DebugMsg( Replicate("-",50))

    ENDIF

RETURN SELF
```

7.1.4. Die Dispatch-Methode

```
METHOD Dispatch( ) AS STRING PASCAL CLASS StdHTTPContext

    DO CASE
    CASE SELF:GetServerVariable ("REQUEST_METHOD") == "GET"
        _DOUT("Method: GET")

        DO CASE

        CASE SELF:CheckParameter ("pane_id")

        OTHERWISE
            // Exe has been called without parameter -> starting point.
            *SELF:OpenLoginForm ()

        ENDCASE

    CASE SELF:GetServerVariable ("REQUEST_METHOD") == "POST"
        _DOUT("Method: POST")

        DO CASE
            // ----------
            // Login Form
            // ----------
        CASE SELF:CheckParameter ("pb_search")
            _DOUT("pb_search")
            _DOUT(SELF:aArgs)

            _cResponse := SELF:HTTPHeader()
            _cResponse += "<html>" + LF + LF

            _cResponse += "<h1> Eingegebene Parameter</h1>" + LF

            _cResponse += "<p>" + LF
            _cResponse += "DHL-Referenz:"
                _cResponse += SELF:aArgs[1][2]
            _cResponse += "</p>" + LF

            _cResponse += "<p>" + LF
            _cResponse += "LOC Dossier no:"
                _cResponse += SELF:aArgs[2][2]

            _cResponse += "</p>" + LF
            _cResponse += "</html>" + CRLF + CRLF

        CASE SELF:CheckParameter ("pb_vsariablen")
            _DOUT("pb_variablen")
            _cResponse := SELF:HTTPRequestItems()

        OTHERWISE
            _DOUT("unbekannt")
            _cResponse := SELF:HTTPHeader()
            _cResponse += "<html>" + LF + LF
            _cResponse += "<h1> Unbekannter Befehl</h1>" + LF
            _cResponse += "</html>" + CRLF + CRLF

        ENDCASE

    OTHERWISE
        _DOUT("Weder GET noch POST")
        _cResponse := SELF:HTTPRequestItems()

    ENDCASE

    RETURN _cResponse
```